CDS-130/02 SPRING 2012: COMPUTATIONAL PROJECT
DUE IN CLASS ON 9 MAY 2012

DETECTING ISOLATED BLACK PIXELS IN A 10x10 SQUARE IMAGE

Introduction:

Detection of isolated pixels in images is a very important image processing technique that finds wide application in science and industry. Your task, in this computational project, is to create an algorithm that detects and counts isolated black pixels in a black-and-white image and then implement that algorithm in Matlab code. Black-and-white images are represented as matrices (called "A") that contain just two values: 1 ("black") and 0 ("white").

You'll want to review the image processing logic we discussed in class. You may also want to refer to past assigned homework problems for additional guidance.

Problem statement:

Given an input image matrix A containing only 1's and 0's, where 1 represents the color black and 0 represents the color white, design and successfully implement an algorithm in Matlab that **detects and counts isolated black pixels** in A. After your program is finished processing A, it will output a single number that is the count of the total number of isolated black pixels detected in A. NOTE: The phrases and words, "matrix A", "A", "image", "the image", etc., will be used interchangeably throughout this document. They are synonymous with the image matrix A.

Algorithm planning:

You are provided shell Matlab code that you must complete in order to accomplish the image processing task described in the above Problem Statement. Please see Appendix #1 for this code. Your contribution will be inserted where you see the second red box: That's where your detection logic, encoded in Matlab statements, will be appear. PLEASE NOTE: **DO NOT** try to copy either red box into your Matlab code! These red boxes are shown **for illustrative purposes only**, to indicate where in the provided Matlab code you must add your own Matlab code.

CDS-130/02 SPRING 2012: COMPUTATIONAL PROJECT
DUE IN CLASS ON 9 MAY 2012

The input image will be a 10x10 matrix possessing 10 rows and 10 columns. Each matrix element will contain either the value 1 or the value 0. No other values are permitted in matrix A. So your detector logic must only look for 1 and 0.

Algorithmic design questions that <u>should be considered</u> prior to writing the first line of Matlab code:

In order to write Matlab code to solve this problem, you'll need to ask yourself and answer the following questions – otherwise, you won't have a clear picture of what you're trying to do and thus, your Matlab code won't accomplish the required task:

1. **What's an isolated black pixel?** How is "isolated" defined (See Appendix 2)? Black and white were already been defined for you: 1 is "black" and 0 is "white". These definitions, when combined with the definitions for "isolated", will govern how your detector logic should be written. Note that where a pixel is located in the matrix, and whether or not it has neighbors (and the neighbors' color!) define whether or not the pixel is isolated! We are thus led to ask . . .
2. **Do all isolated black pixels look the same?** How do isolated black pixels ***<u>appear</u>*** at various locations inside input matrix A? Again, refer to Appendix 2. For example, does an isolated black pixel in the center of matrix A look the same as an isolated black pixel on the edge of A, or in one of the four corners of A? Do isolated black pixels in the four corners of A all look the same? Differences in appearance will tell you exactly how detection logic should be written, and thus, guide your algorithm design and your Matlab implementation.
3. **How should my program distinguish between the image corners, edges and interior?** Understanding matrix A's boundaries is vitally important for two reasons: (a) The definition of an isolated black pixel depends upon where it's located in A; and (b) You don't want your detector logic to "fall off" any edge of A as it processes the matrix! As you know, when a program "falls off the edge" of a matrix, it causes Matlab to terminate further processing and generate an error message. Meaning: "falling off the edge" of A will prevent your program from earning all possible points!
4. **How can my program keep a running count of isolated black pixels?** Of course, each time your program detects an isolated black pixel, it must record this fact. After image processing is complete, your program will generate a single numerical value that represents the sum total of all isolated black pixels it finds. (HINT: Can you keep track of the number of times that IF statement branches are entered? Sure you can! Simply

include the statement "isolated_black_pixels = isolated_black_pixels + 1" inside the branches of interest!)

Algorithmic design questions that <u>should be considered</u> prior to submitting your project for grading:

In order to gain the highest possible scores on this project, you should ask and answer these questions for yourself:

1. **What does my Matlab program report for "weird" input cases?** This is a key question to answer, precisely because your program may be provided "weird" inputs to process. What's an example of a "weird" input? Well, how about an image consisting of the value 1 **in every element A(m,n) of input matrix A**? That is, A contains all 1's and no 0's (in other words, the image is pure black). How many isolated black pixels would your program report for that particular input? The answer, of course, is that your program must report **<u>ZERO ISOLATED BLACK PIXELS</u>** for that particular input: When the entire input image is black, there are no isolated black pixels – by definition! Does your program correctly report this result? If not, then your program has a bug in it, and you'll need to work through your detector logic to discover why your program computes the incorrect answer for a pure black image. The same holds true for a pure white image (consisting of all 0's).
2. **Does my program produce the correct answer for all the test cases I try?** If not, then again, your program has a bug in it and you need to work carefully through the detector logic, for each cell of the input matrix. **<u>Pay particular attention to those inputs that your program detects and counts correctly, and those it detects and counts incorrectly</u>**. It's very likely when inspecting specific inputs corresponding, respectively, to correct and incorrect counts, that you'll notice a particular structural feature that differentiates the inputs. That's excellent information to have, because then you can go to the precise location in your program where that particular structural feature is (or should be) detected, and, inspect that precise location for possible bugs. **<u>ALWAYS</u>** pay attention to incorrect outputs, and, to the corresponding inputs that produced these incorrect outputs! These are very informative failures!

Creating your Matlab code:

Please refer to the Matlab code shell, called "detectorshell.m" found in the Appendix #1 and distributed separately. To complete the required coding, you must write code to "fill in" the

spaces indicated by the two red boxes. **Do not attempt to copy these red boxes into your actual Matlab code!** Rather, just recognize that these are placeholders telling you where your code should be inserted into the detectorshell.m program. Please note that the size of the red boxes does not indicate the amount of code that you must write. For example, the first red box represents a single Matlab statement, whereas the second red box represents several Matlab statements (HINT: The code for the second red box will consist of an IF/ELSEIF/…/ELSEIF/END construction inside a double nested FOR loop).

Testing your Matlab code:

It's a VERY GOOD IDEA to test your Matlab program before submitting it for grading. At a minimum, you should try to reproduce the test results shown in Appendix #3. If you can do that, you should have good isolated black pixel detectors written, and these are probably operating properly. To gain confidence that this is the case, you should also create your own test cases to process. Remember: Reproducing Appendix #2 test results means that the title on the image produced by your code, and the number it reports, MUST correspond to the **CORRECT** number of isolated black pixels in the image! It's not enough to simply create an image of the test matrix A but report the incorrect isolated black pixel count! Doing so will lose points.

What do I need to turn in for this project?

Very simply, your properly commented, well-structured, correctly written Matlab code. Nothing more. **I would like BOTH hard and soft copies**: Hard copy handed in to me during our final class meeting on 2 May, and soft copy emailed to me, also during class – there's no excuse for giving me a hard copy and not emailing me the corresponding soft copy (or vice versa). Also please ensure that the code you email me is the same one that you hand in to me in hardcopy! If it isn't, I won't grade it.

CDS-130/02 SPRING 2012: COMPUTATIONAL PROJECT
DUE IN CLASS ON 9 MAY 2012

How will this project be graded?

This assignment consists of two pieces, homework #10 and homework #11, each worth 20 points. The first piece, homework #10, will focus on evaluating your Matlab code. Specifically:

1. Is your Matlab code well written, using proper Matlab syntax?
2. Does your Matlab code actually solve the problem, that is, does it actually count isolated black pixels in an input matrix A?
3. Is your Matlab code COMMENTED appropriately? Meaning: Does it include additional comments where you've added code to that provided in Appendix #1? Are you listed as an author?

The second piece (20 points; corresponding to homework #11) will focus on evaluating the **performance** of your Matlab code. This is necessarily a more stringent evaluation, and represents the first time in this class (and the only time) that your code will be evaluated on how well it performs a task, and not just whether it can perform that task at all. This time, you'll need to be substantially closer to the correct answer than just "nearby". You will need to be exactly correct.

The evaluation criterion is simple: I will present your code with approximately 45,000 test images. Points earned on homework #11 will depend on how many of those images your Matlab code processes correctly. Meaning: For a specific input image matrix A, does your code correctly count and report all the isolated black pixels in that image? The more input matrix images your code processes correctly, the more points you will earn, up to a total of 20 points. **Please note**: I will NOT, under any circumstances, alter your code before or during testing! What this means is that the code you send me is the code that I will test. If I notice errors in your code prior to testing, well, that's the code that you sent to me and so that's the code that I will test, errors and all. So please ensure that the code you want tested is the code that you send me. Since this is a homework assignment, the late homework policy is in effect: I won't accept late programs or "second versions" of programs after 11:45am on 9 May 2012.

Since this assignment is comprised of two combined homeworks, the drop policy is in effect for each separate piece: Each will be entered as a separate homework assignment grade, each out of 20 possible points. At the end of the semester, I'll retain your highest ten homework grades out of the eleven, when computing your final course grade.

CDS-130/02 SPRING 2012: COMPUTATIONAL PROJECT
DUE IN CLASS ON 9 MAY 2012

APPENDIX #1 – MATLAB CODE: detectorshell.m

```
%PROGRAM:  detectorshell.m
%AUTHOR:  ******** YOUR NAME GOES HERE ********
%DATE OF LAST MODIFICATION:  ***** DATE YOU FINISH GOES HERE  *****

%PURPOSE: Homework assignments #10 and #11 for CDS-130 Spring 2012.


%CHANGELOG:

%DATE          MODIFICATION                                PROGRAMMER
%----------    ------------------------------------   ----------
%XX XXX 2012   XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX       YOU



%VARIABLES LIST (ARRANGED ALPHABETICALLY):
%----------------------------------------

% A:  The image matrix consisting of 1's ("black") and 0's ("white")
% isolated_black_pixels:  running count of isolated black pixels in A
% mymap:  Color map defined to displaying 1 as black and 0 as white
% totalrows:  The total number of rows in the matrix A
% titlestr:  String that defines what is written as the image's title
% totalcols:  The total number of columns in the matrix A


clear;clc

%---------- INPUT PARAMETER DEFINITIONS -------------------------------

%define a 10x10 image matrix A

A = [0 0 0 0 1 0 0 0 0 0;
     0 0 1 1 1 0 0 0 0 0;
     0 0 0 0 0 0 0 1 1 0;
     0 0 0 0 0 0 0 1 0 0;
     0 0 0 0 0 0 0 0 0 0;
     0 0 1 0 0 0 0 0 0 0;
     0 0 1 0 0 0 0 1 1 1;
     0 0 0 0 0 0 0 0 0 0;
     1 0 0 0 1 1 0 0 0 0;
     1 1 0 0 0 0 0 0 0 0];


%declare a variable called "isolated_black_pixels", and initialize it to 0:
```

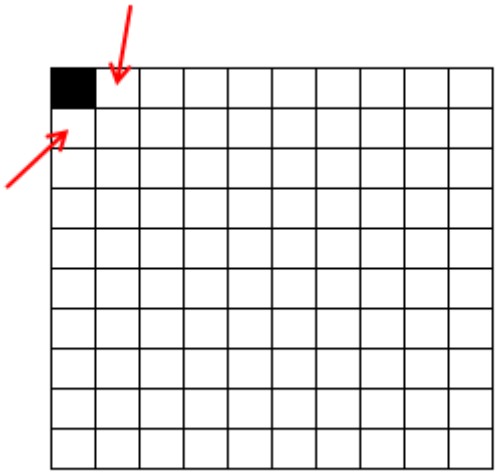
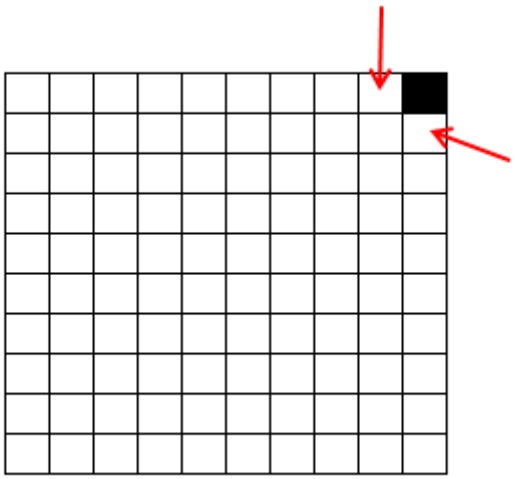
CDS-130/02 SPRING 2012: COMPUTATIONAL PROJECT
DUE IN CLASS ON 9 MAY 2012

```
%---------- IMAGE PROCESSING ----------------------------------------

%Your detector code goes here:
```

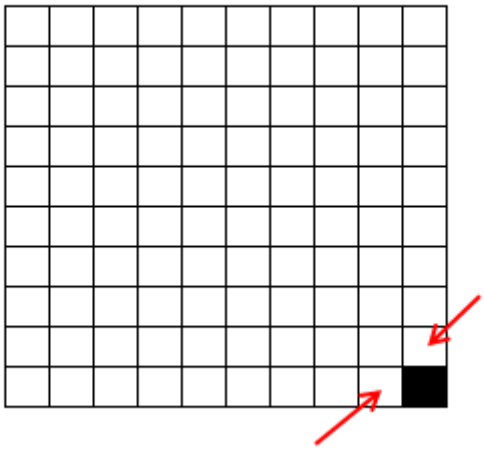
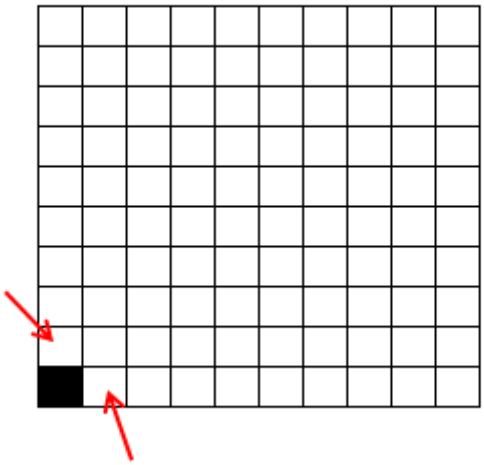
```
%report total number of isolated black pixels detected:
isolated_black_pixels


%---------- IMAGE POST-PROCESSING/VISUALIZATION -----------------------

%define the color map
mymap = [1 1 1;
         0 0 0];

%set the Matlab system color map to the one just defined
colormap(mymap);

%put the image of matrix A up on screen
imagesc(A)

%define the title that will be plotted on the image
titlestr = horzcat('Total Isolated Black Pixels in Below Image: ',...
   int2str(isolated_black_pixels));

%put the title on the image plot
title(titlestr,'FontName', 'Arial', 'FontSize', 14,'FontWeight', 'Bold')
axis square
set(gca,'XTickLabel',[])
set(gca,'YTickLabel',[])
```

CDS-130/02 SPRING 2012: COMPUTATIONAL PROJECT
DUE IN CLASS ON 9 MAY 2012

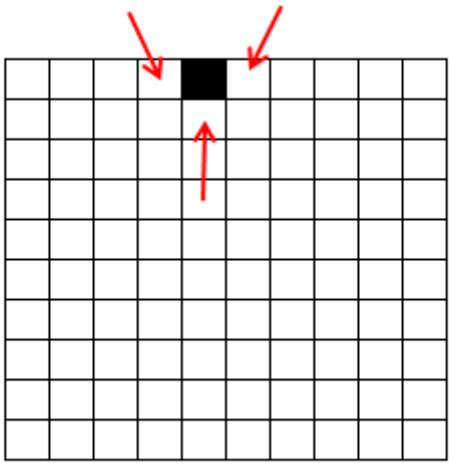
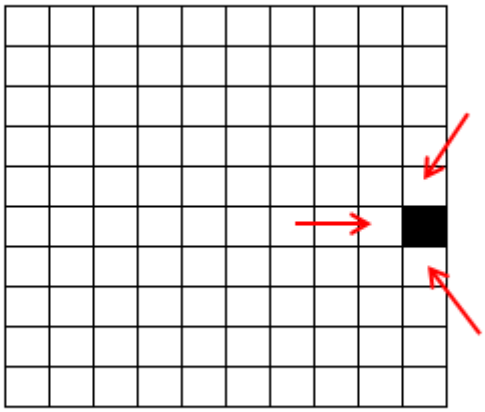
APPENDIX #2 – DEFINITIONS OF ISOLATED BLACK PIXELS, WITH EXAMPLES

Isolated Pixel Definition	Explanation
	**NORTHWEST CORNER**: In this location, a black pixel is isolated if the pixels directly to the right and directly below (as indicated by the red arrows) are white. Note that **three pixels** are examined.
	**NORTHEAST CORNER**: Here, a black pixel is isolated if the pixels directly to the left and directly below (as indicated by the red arrows) are white. Note that three pixels are examined.

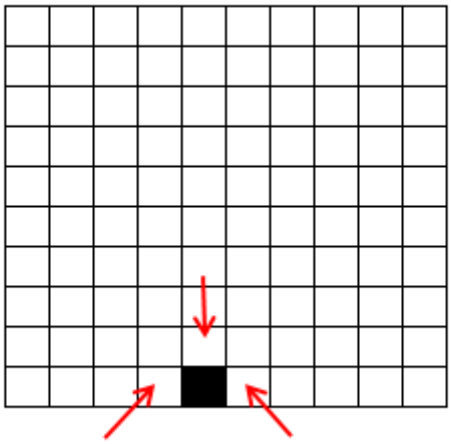
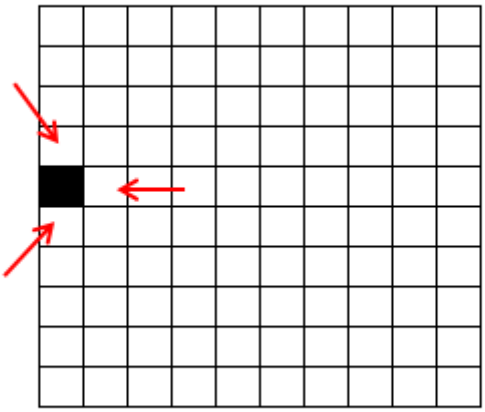
CDS-130/02 SPRING 2012: COMPUTATIONAL PROJECT
DUE IN CLASS ON 9 MAY 2012

	**SOUTHEAST CORNER**: Black pixels in the southeast corner are isolated if the pixels directly to the left and directly above (as indicated by the red arrows) are white. Note that three pixels are examined.
	**SOUTHWEST CORNER**: A black pixel located in the southwest corner is isolated if the pixels directly to the right and directly above (as indicated by the red arrows) are white. Note that three pixels are examined.

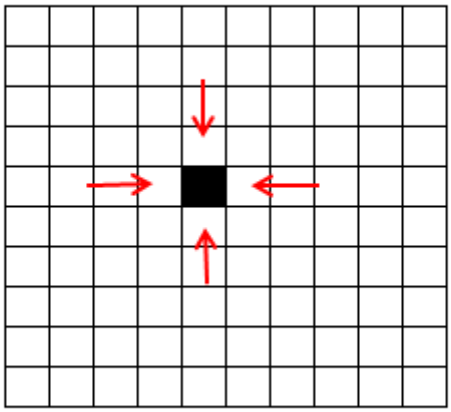
CDS-130/02 SPRING 2012: COMPUTATIONAL PROJECT
DUE IN CLASS ON 9 MAY 2012

	**NORTH EDGE**: Black pixels located on the north edge are isolated if the pixels directly to the left, to the right and below (as indicated by the red arrows) are white. Note that **four pixels** are examined.
	**EAST EDGE**: Black pixels located on the east edge are isolated if the pixels directly to the left, above and below (as indicated by the red arrows) are white. Note that four pixels are examined.

CDS-130/02 SPRING 2012: COMPUTATIONAL PROJECT
DUE IN CLASS ON 9 MAY 2012

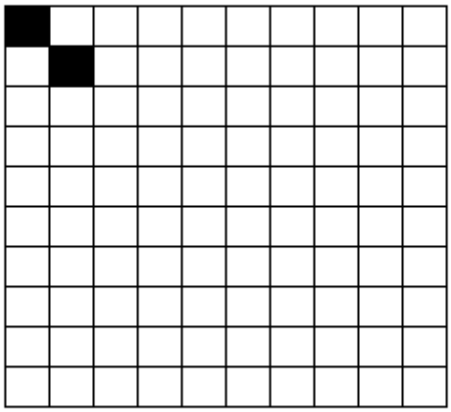
	**SOUTH EDGE**: A black pixel located on the south edge is isolated if the pixels directly to the left, to the right and above (as indicated with by red arrows) are white. Note that four pixels are examined.
	**WEST EDGE**: Black pixels located here are isolated if the pixels directly above, below and to the right (as indicated by the red arrows) are white. Note that four pixels are examined.

CDS-130/02 SPRING 2012: COMPUTATIONAL PROJECT
DUE IN CLASS ON 9 MAY 2012

	**CENTER**: Pixels located in the center of the image are isolated if the pixels directly above, below, to the right and to the left (as indicated by the red arrows) are white. Note that **five pixels** are examined.

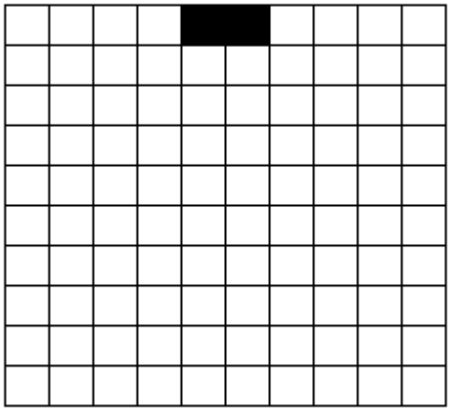
CDS-130/02 SPRING 2012: COMPUTATIONAL PROJECT
DUE IN CLASS ON 9 MAY 2012

Some Examples With Answers



Q: How many isolated black pixels?

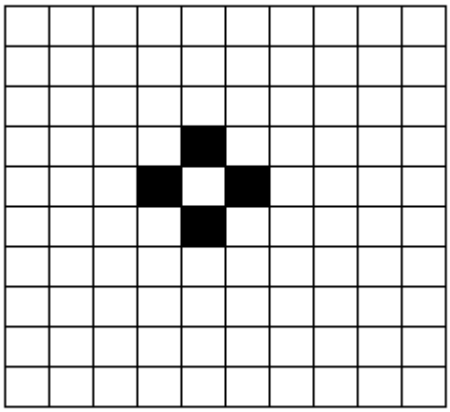
A: 2



Q: How many isolated black pixels?

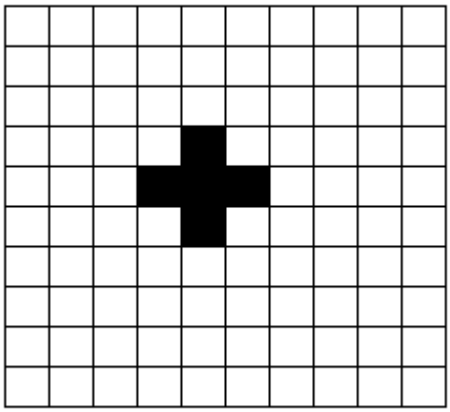
A: 0

CDS-130/02 SPRING 2012: COMPUTATIONAL PROJECT
DUE IN CLASS ON 9 MAY 2012



Q: How many isolated black pixels?

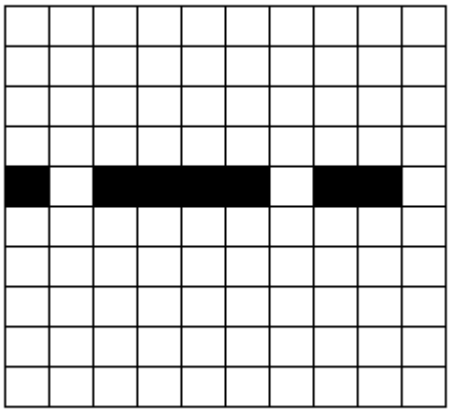
A: 4



Q: How many isolated black pixels?

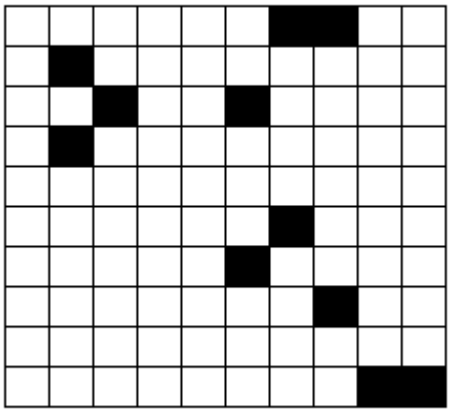
A: 0

CDS-130/02 SPRING 2012: COMPUTATIONAL PROJECT
DUE IN CLASS ON 9 MAY 2012



Q: How many isolated black pixels?

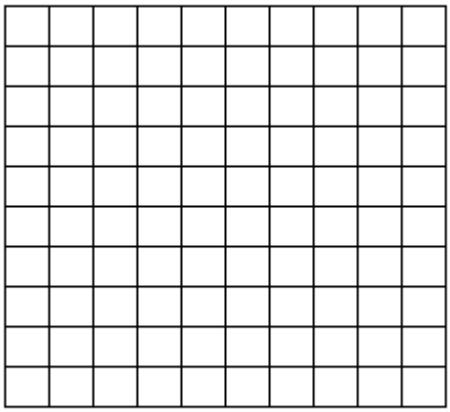
A: 1



Q: How many isolated black pixels?

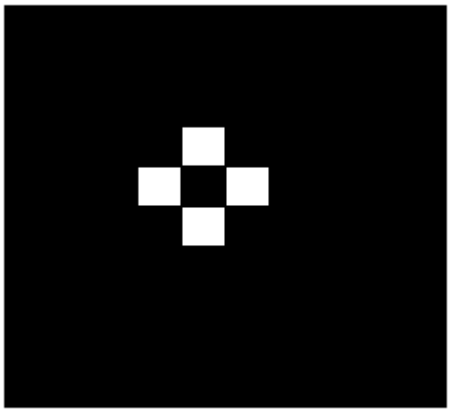
A: 7

CDS-130/02 SPRING 2012: COMPUTATIONAL PROJECT
DUE IN CLASS ON 9 MAY 2012



Q: How many isolated black pixels?

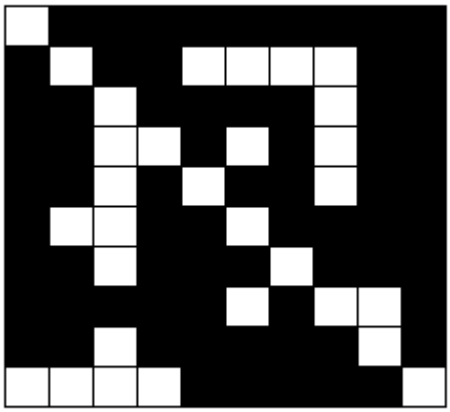
A: 0



Q: How many isolated black pixels?

A: 1

CDS-130/02 SPRING 2012: COMPUTATIONAL PROJECT
DUE IN CLASS ON 9 MAY 2012



Q: How many isolated black pixels?

A: 0

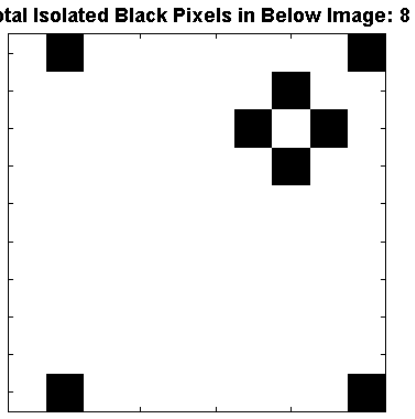
CDS-130/02 SPRING 2012: COMPUTATIONAL PROJECT
DUE IN CLASS ON 9 MAY 2012

APPENDIX #3 – EXAMPLES OF SUCCESSFUL DETECTIONS AND COUNTS

Each of the following input matrices A produces the corresponding output graphic and isolated black pixel count. These samples are designed to show you what output you should obtain from your algorithm, **FOR THE EXACT SAME INPUT A** that is listed to the left of the graphic. Thus, consider these examples to be test cases: If your code's output graphic, for the same input A, does not match the graphic seen to the right of matrix A in the table below, then you should inspect your code and try to figure out why your output differs. Note the title on each graphic: This title reports the total count of isolated black pixels in the image.

To evaluate your code's performance, I'll execute it about 45,000 times, each time with a different input A. Your code's homework #11 performance grade will depend upon how many inputs your code processes correctly: For how many of the approximately 45,000 test images does your code report correctly the total number of isolated black pixels?

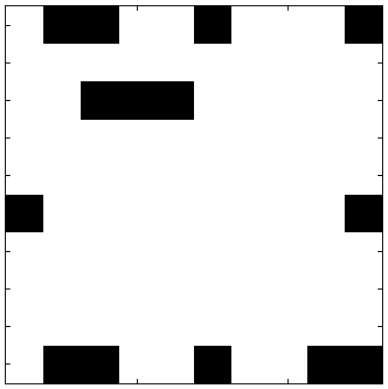
Sample Input Image Matrix A	Corresponding Output and Count

`A = [0 1 0 0 0 0 0 0 0 1;`
`0 0 0 0 0 0 0 1 0 0;`
`0 0 0 0 0 0 1 0 1 0;`
`0 0 0 0 0 0 0 1 0 0;`
`0 0 0 0 0 0 0 0 0 0;`
`0 0 0 0 0 0 0 0 0 0;`
`0 0 0 0 0 0 0 0 0 0;`
`0 0 0 0 0 0 0 0 0 0;`
`0 0 0 0 0 0 0 0 0 0;`
`0 1 0 0 0 0 0 0 0 1];`
	

CDS-130/02 SPRING 2012: COMPUTATIONAL PROJECT
DUE IN CLASS ON 9 MAY 2012

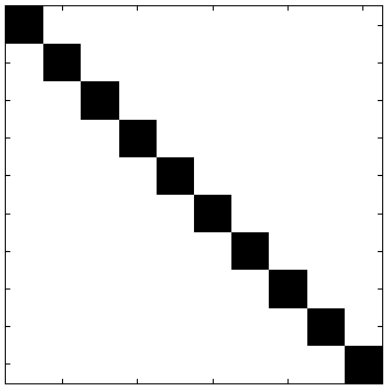
```
A = [0 1 1 0 0 1 0 0 0 1;
     0 0 0 0 0 0 0 0 0 0;
     0 0 1 1 1 0 0 0 0 0;
     0 0 0 0 0 0 0 0 0 0;
     0 0 0 0 0 0 0 0 0 0;
     1 0 0 0 0 0 0 0 0 1;
     0 0 0 0 0 0 0 0 0 0;
     0 0 0 0 0 0 0 0 0 0;
     0 0 0 0 0 0 0 0 0 0;
     0 1 1 0 0 1 0 0 1 1];
```

Total Isolated Black Pixels in Below Image: 5



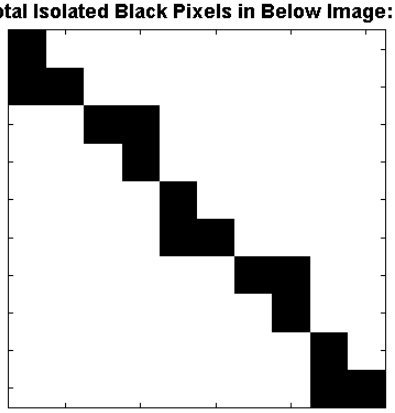
```
A = [1 0 0 0 0 0 0 0 0 0;
     0 1 0 0 0 0 0 0 0 0;
     0 0 1 0 0 0 0 0 0 0;
     0 0 0 1 0 0 0 0 0 0;
     0 0 0 0 1 0 0 0 0 0;
     0 0 0 0 0 1 0 0 0 0;
     0 0 0 0 0 0 1 0 0 0;
     0 0 0 0 0 0 0 1 0 0;
     0 0 0 0 0 0 0 0 1 0;
     0 0 0 0 0 0 0 0 0 1];
```

Total Isolated Black Pixels in Below Image: 10



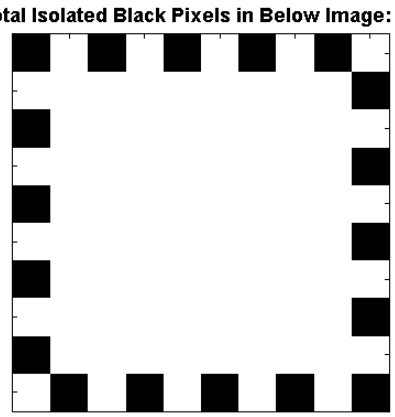
```
A = [1 0 0 0 0 0 0 0 0 0;
     1 1 0 0 0 0 0 0 0 0;
     0 0 1 1 0 0 0 0 0 0;
     0 0 0 1 0 0 0 0 0 0;
     0 0 0 0 1 0 0 0 0 0;
     0 0 0 0 1 1 0 0 0 0;
     0 0 0 0 0 0 1 1 0 0;
     0 0 0 0 0 0 0 1 0 0;
     0 0 0 0 0 0 0 0 1 0;
     0 0 0 0 0 0 0 0 1 1];
```

Total Isolated Black Pixels in Below Image: 0

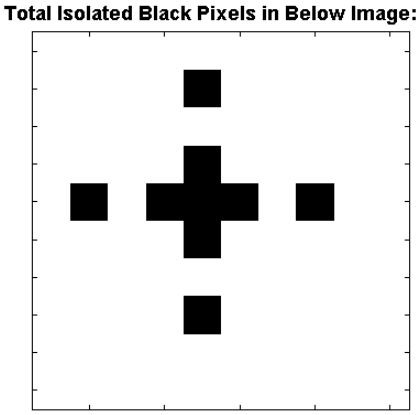


CDS-130/02 SPRING 2012: COMPUTATIONAL PROJECT
DUE IN CLASS ON 9 MAY 2012

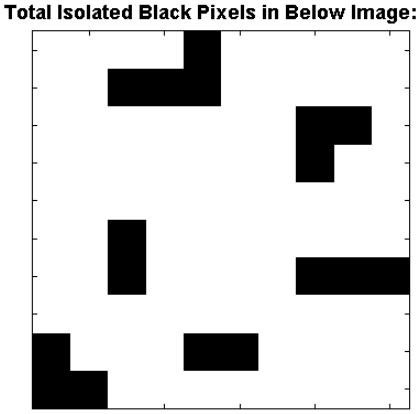
```
A = [1 0 1 0 1 0 1 0 1 0;
     0 0 0 0 0 0 0 0 0 1;
     1 0 0 0 0 0 0 0 0 0;
     0 0 0 0 0 0 0 0 0 1;
     1 0 0 0 0 0 0 0 0 0;
     0 0 0 0 0 0 0 0 0 1;
     1 0 0 0 0 0 0 0 0 0;
     0 0 0 0 0 0 0 0 0 1;
     1 0 0 0 0 0 0 0 0 0;
     0 1 0 1 0 1 0 1 0 1];
```




```
A = [0 0 0 0 0 0 0 0 0 0;
     0 0 0 0 1 0 0 0 0 0;
     0 0 0 0 0 0 0 0 0 0;
     0 0 0 0 1 0 0 0 0 0;
     0 1 0 1 1 1 0 1 0 0;
     0 0 0 0 1 0 0 0 0 0;
     0 0 0 0 0 0 0 0 0 0;
     0 0 0 0 1 0 0 0 0 0;
     0 0 0 0 0 0 0 0 0 0;
     0 0 0 0 0 0 0 0 0 0];
```




```
A = [0 0 0 0 1 0 0 0 0 0;
     0 0 1 1 1 0 0 0 0 0;
     0 0 0 0 0 0 0 0 1 1 0;
     0 0 0 0 0 0 0 0 1 0 0;
     0 0 0 0 0 0 0 0 0 0 0;
     0 0 1 0 0 0 0 0 0 0 0;
     0 0 1 0 0 0 0 1 1 1;
     0 0 0 0 0 0 0 0 0 0 0;
     1 0 0 0 1 1 0 0 0 0;
     1 1 0 0 0 0 0 0 0 0];
```



You should execute your code on these test cases – copy the matrix A shown in the left column and paste it into your Matlab code, at the front, right where matrix A is located in the code.

Then, when you run your code with this input test case, your code should produce the exact graphic seen on the right, along with the exact count of isolated black pixels shown in the image's title. You can create your own test images by rearranging the 1's and 0's inside the matrix A, running your code on this new test matrix, and observing whether you obtain the correct output. These "self-generated" test cases will be very useful to you because you'll be able to see whether your detector algorithm works as advertised: you'll be able to visually count the number of isolated black pixels in the output, and compare to the count your code produces.